Blobswarm

Operativsystem och mutlicoreprogrammering (1DT089)

Projektrapport grupp 14: Andreas Gäwerth (920211-2272), Anton Carlsson (920608-5616), Martin Källström (920127-1898), Richard Stråhle (910101-4794), Tomas Ringefelt (820320-2554)

Version 2, 2014-06-07

Innehållsförteckning

1. Inledning
2. Översikt över systemet
    2.1 Systemdesign
3. Implementation
4. Slutsatser
Appendix: Installation och utveckling

1. Inledning

Att vara jagad kan både vara det bästa som finns eller det värsta som kan hända en. Kunskap, utseénde eller karisma kan vara några av de saker som gör att man blir jagad på det “bra” sättet. Folk vill ha/se/veta mera om dig, vilket kan vara både roligt eller jobbigt. Detta kallar jag att vara psykiskt jagad.

Ett annat sätt att vara jagad är att vara fysiskt jagad. Folk eller kanske till och med varelser av alla olika slag är ute efter dig. Vad gör man i en sådan situation? Springer man iväg? Gömmer sig i ett hörn? Eller kanske kämpar för sitt liv. Alla är vi olika och vad vi gör beror mycket på allvaret i situationen och vilka möjligheter vi har.

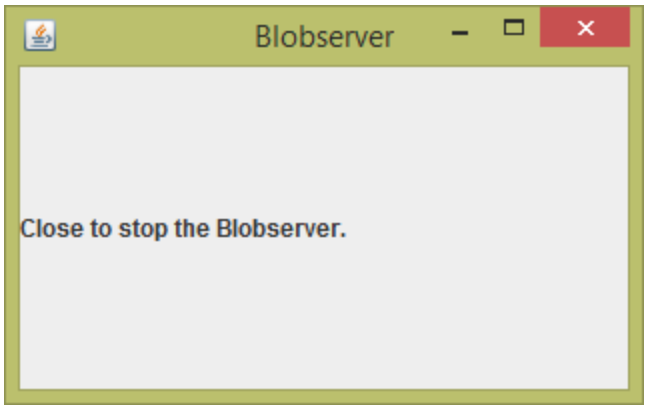
Vi ville skapa ett spel där man kanske får uppleva en liten del av den känslan och skapade Blobswarm. Samtidigt ville vi lära oss utnyttja concurrency på ett sätt som var relevant för framtida projekt i arbetslivet. De sätt vi använde oss utav concurrency kändes perfekt för projektet. Vi programmerade så att varje datorstyrd varelse (NPB, Non-Player Blob) kördes på en egen tråd som en egen självständig individ och kan därmed göra sin egna tolkning av vad den ska göra.

Blobswarm är ett multiplayerspel där spelarens uppgift är att överleva så länge som möjligt samtidigt som man jagas av datorstyrda varelser(Blobar). De datorstyrda blobarna “jagar” i grupp och och kommer alltså försöka hålla ihop som en svärm när de jagar spelare. Den spelare som överlever längst vinner.

2. Översikt över systemet

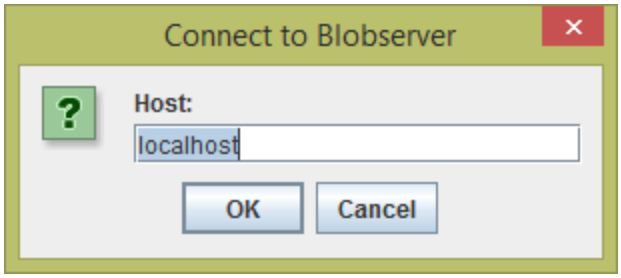
Blobswarm består av två delar: Server och Client.

Server

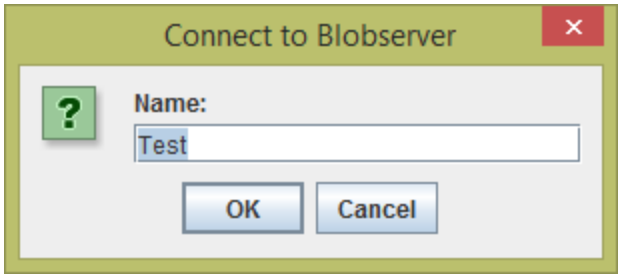


Servern har en väldigt simpel GUI. Det är en frame som säger användaren att servern är online, och när man stänger den är servern inte online längre.

Client



Client har också en simpel GUI. När man startar clienten får man skriva in vilken server man vill ansluta sig till, med standardinput som lokalt, alltså att servern körs på samma dator som clienten, annars får man skriva in IP:n till den dator som kör servern man vill ansluta sig till.

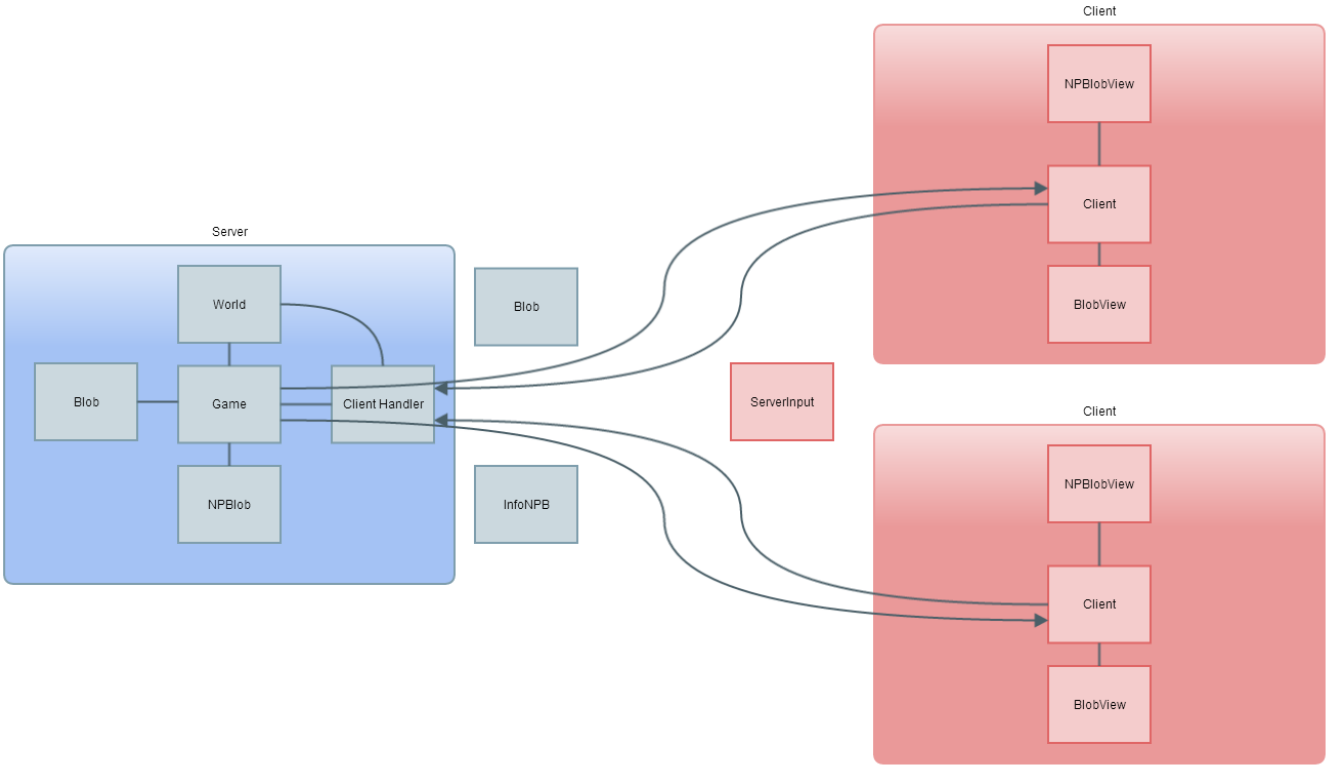


Efter detta skriver man in ett namn, och sedan är man inne i spelet.

Man ser då en grön bakgrund och ett antal blobs, varav en är sin egen man kan styra med W, A, S och D. För att sluta spela stänger man helt enkelt clienten. Röda blobs är styrda av servern, medan gröna blobs är spelare.

2.1 Systemdesign

Systemarkitektur



Först och främst är systemet uppdelat i Server och Client.

Client tar hand om att ta input från en användare som den skickar till en server och att rita ut en grafisk representation av information den får från servern.
Client har information om de grafiska representationerna av blobs i form av BlobView för clientkontrollerade blobs och NPBlobView för serverkontrollerade blobs.

Inom serverdelen av systemet har vi en central del som vi kallar Game, som skickar information om alla blobs till clienterna i form av två arrayer. En array av Blob som innehåller all information om clientkontrollerade blobs och en array av InfoNPB som innehåller positioner och direktioner av alla serverkontrollerade blobs.
World hanterar positioner av alla blobs och flyttningar av dessa blobs.
ClientHandler tar emot och skickar vidare input från Clients till World.

Concurrency
Vi använder trådar för att hantera alla serverkontrollerade blobs (NPB:er). Serverprocessen skapar en ny tråd varje gång en NPB skapas och den tråden hanterar den NPB:ns beräkningar, vilket innebär att alla NPB:ers beräkningar och rörelser sker concurrent. Synkronisering behövs ej här för en NPB försöker endast röra på sig själv, inte på någon annan.

Servern som haterar själva värden har även en flytta funktion som endast en får använda sig av åt gången som ett mutex lås. Detta för att se till när en eller flera spelar vill gå samtidigt så tillåts endast en gå så de inte uppstår olika buggar tack vara datarace som kan uppstå.

3. Implementation

3.1 Språk
Systemet är implementerat i Java, vilket vi valde för att alla i gruppen testat och känner sig bekväma med hur man skriver kod i Java.
Fördelar med att skriva vårt projektet i Java är man får mycket gratis. Java har funnits länge och det finns därför nästan oändligt många bibliotek. Det medför även att det är lätt att hitta information om problem som många andra haft tidigare, få bra förklarande exempel på funktioner eller hur de kan kombineras. Java som språk är även lätt att förstå, har bra struktur och har bra verktyg att tillgå som dessutom har liknande syntax som andra stora språk. Men det bästa för oss med Java är att det finns bra verktyg/metoder som är lätta att använda när man ska implementera concurrency.

3.2 Datastrukturer och algoritmer
Blob
Blob är en central datastruktur för systemet. Den innehåller en position, ett ID, en riktning, ett namn, en hastighet, och två booleans som anger om denna Blob lever eller inte samt om den är osårbar. En viktig algoritm för en Blob är contains. Contains tar en Point och returnerar true om argumentet ligger den del av spelytan Blob:en tar upp, annars false. Detta används för att kolla collision. En annan viktig algoritm är updateLocation, som tar en int som argument. Beroende på vad denna int är flyttas Blob:en åt en riktning, Blob:ens hastighet avgör hur mycket åt det hållet den rör sig.

NPBlob
NPBlob är en annan central datastruktur för systemet som ärver av Blob.
Förutom det som NPBlob ärver har en NPBlob en aggression som avgör hur aggressiv denna NPBlob är mot spelarkontrollerade Blob:ar, en array av Blob:ar som innehåller alla dessa, ett mål som är den Blob som NPBlob:en vill röra sig emot, en lista av NPblob:ar för att veta var övriga NPBlob:ar är, och en hastighetsbegränsning.

NPBlobarnas AI är baserad på en s.k swarming algoritm. Den är skapad av Craig Reynolds 1986 för att simulera flockande fåglar, och kallades boids. Algoritmen har senare använts i en rad sammanhang i både datorspel och animationer.

Algoritmen fungerar så att tre krafter räknas ut och adderas till accelerationen.
Krafterna är:

Separation styr för att undvika flockmedlemmar som är för nära varandra.
Cohesion styr
Alignment styr mot den genomsnittliga riktningen för de närmaste flockmedlemmarna.
En annan viktig algoritm för en NPBlob är getNearestBlob, som returnerar den spelarkontrollerade Blob som är närmast denna NPBlob. Detta används för att välja ett mål.

ServerInput
Client skickar en datastruktur som heter ServerInput till servern den är ansluten till. Den innehåller 4 booleans, false på de riktningar där den inte går och true om den går åt den riktningen.

NPblobs och Blobs
Servern skickar hela tiden ut 2 datastrukturer som heter NPBlobs och Blobs till alla clienter. En innehåller en array av InfoNPB. Varje InfoNPB har data om en NPB som berättar var den befinner sig och vilket håll den ska titta åt (grafikmässigt). Den andra arrayen är spelarnas Blob:ar som servern uppdaterat med saker som hänt och skickar tillbaka hela Bloben som består av diverse data som om den fortfarande lever, sin nya position, ID, namn osv.

updateNPBs
updateNPBs är en algoritm som körs av clienten varje gång den tar emot information om NPBlob:ar från servern. Algoritmen uppdaterar alla clientens grafiska representationer av NPBlob:ar med informationen från de InfoNPB:er den fick från servern.
updateBlobs
updateBlobs är den algoritm som körs av clienten när den tar emot information om Blob:ar från servern. Den matchar ID:s mellan de grafiska representationerna av Blob:ar som clienten har och de Blob:ar som den fick från servern, och sedan uppdateras de grafiska Blob:arna med den nya informationen så att de har rätt position, riktning, och livstillstånd.
checkNewBlobs

checkNewBlobs körs innan updateBlobs och kollar ifall det finns en Blob bland informationen från servern som inte kan matchas mot en grafisk Blob. Isåfall skapas en ny för denna Blob.
removeDeadBlobs
removeDeadBlobs körs också innan updateBlobs för att kolla ifall det finns en grafisk Blob på clienten som inte längre finns bland informationen från servern. Detta innebär att den client som styr den Blob:en inte längre spelar, och då tas den grafiska representationen för den Blob:en bort.

3.3 Concurrency

Vår modell till concurrency har fördelen att alla NPB är självständiga vilket betyder man kan enkelt dela upp NPBerna att göra vad de vill. Vi valde att göra detta för att varje NPB behöver räkna ut cohesion, alignment, och separation och om detta händer i trådar kan alla NPBs göra det samtidigt vilket ger oss concurrency.

Deadlocks har undvikits och kan inte uppstå i våran kod. Detta för att NPB:erna endast låser arraylistan av NPB:er för att uppdatera sig själv och låser upp efter att den uppdaterat sin egen position. På alla andra ställen behöver den inget lås när den bara behöver läsa in information för att beräkna ut sin nya position. Så inga deadlocks kan uppstå därför alla kritierier för att deadlock ska kunna uppstå är inte uppfyllda.

3.4 Kryonet

Vi använder oss av ett API kallat Kryonet för nätverkskommunikationen. Kryonet gör hantering av nätverkskommunikation mycket enklare då det fungerar som ett lager ovanpå tcp/ip i Java.

Kryonet består av två komponenter.
1. Kryo serializer
2. Kryonet server/client

Servern kör på en egen tråd och kan skicka meddelanden asynchront över tcp och udp.
Serializern kan översätta objekt till ett format som kan skickas över Kryonetkanalen.
När man vill skicka ett objekt måste klassen först registreras i Serializern. Sedan är det väldigt lätt att skicka och ta emot objekt.

Kryonet är snabbt, smidigt och lämpar sig väl för just multiplayer spel.

4. Slutsatser

Vårat projekt gick ut på att få en ökad förståelse över hur concurrency fungerar inom java. Att programmera ett spel från grunden med serverdel och clientdel är inte någonting man gör varje dag och kräver såklart många timmars arbete och research.
Att sedan göra detta som grupp har sina fördelar och nackdelar. Fördelarna är såklart att man är flera personer som kan hjälpas åt med uppgiften, man kan dela upp arbetet i olika delar och den som vill/passar bäst till uppgiften kan göra sin del.

Nackdelen är dock densamma… att man är flera personer på samma jobb. Det kan vara väldigt svårt att dela upp ett arbete bra när man inte riktigt vet vad som behöver göras. Det är otroligt lätt att det blir så att några få personer tar på sig/får alldeles för mycket jobb att göra och uppdelningen blir aldrig riktigt bra. Detta kan resultera i att inte hela gruppen har så bra koll på vad som händer och att vissa känner att andra inte gör tillräckligt. Därför är det viktigt med planering, möten och öppenhet inom en grupp.
Att planera vad som ska göras härnäst, vem som ska göra det och hur det skall göras är bra för alla i gruppen. När ska detta då bestämmas? Jo.. på möten där gruppen diskuterar detta och allt annat. Att vara öppen med vad man kan och inte kan är också viktigt så någon inte får jobb den inte klarar av.

Eftersom vi gör ett spel kan det ibland vara lätt att någon också fastnar med att göra mer grafik än kod. Det behövs och är verkligen kul, men det lär en inte så mycket från kursens synvikel, för att spelet ska bli bra är det minst lika viktigt.

Om vi nu skulle göra om projektet eller göra någonting liknande igen kan man fundera på vad som skulle vara annorlunda. Att arbeta för en tydligare uppdelning i arbete är någonting som skulle kunna resultera i ett bättre arbetsflöde.

Förbättringar går alltid att införa i allt man gör. Till vårat spel kan det till exempel vara:
Fler sätt för spelarna att försvara sig. Om man implementerar något vapenliknande i spelet kan spelaren få större chanser att överleva länge. Detta låser även upp möjligheterna att införa PvP (Player vs Player), att ge spelarna möjlighet att attackera varandra eller förstöra för varandra kan ge spelet helt andra sidor.

När det kommer till multiplayerdelen för Blobswarm så skulle en updatering vara att inför någon sorts lobby. Där spelare kan mötas upp innan spelet startar och kanske lägga upp en plan, ställa in inställningar så som storlek på banan, antal NPB:er eller kanske liv.

Möjligheterna för ett sådant spel vi har gjort är i princip oändliga. Vi har skapat en grund och en väldigt simpel spelidé.

Appendix: Installation och utveckling

Vi har använt Java med JRE 1.7 för att implementera vårat system.

Koden finns tillgänglig på GitHub:
https://github.com/ACarlsson2/OSM-grupp-14/tree/master/Blobswarm
JUnit används för automatiserad testning.
Automatiserad generering av dokumentation finns med Javadoc.
Systemet kompileras och startas med Eclipse.

Att göra

“lyfta fram den tekniska biten i vissa stycken”. //Vad för tekniska saker? // Ingen aning, osthyvlar typ